**JVM Architecture:-**

**.**java source file is compiled by Compiler javac-🡪.class file will be created.

This .class file will be given as input to Class Loader subsystem. It consists of three parts

1. Loading
2. Linking
3. Initialization

**Loading:-**It has 3 parts . Extension delegation algorithm.

1. Bootstrap Class loader:- All core java API Classes will be taken care by Bootstrap Class Loader.
2. Extension class Loader:- jdk,ext, lib
3. Application Class Loader:- Classes from applications class path.

**Linking:-** It performs 3 activities. It checks the virus valid or not .

1. Verify -It checks the bytecode valid or not.
2. Prepare:- For static variables memory will be allocated and assigned with Default values.
3. Resolve:-

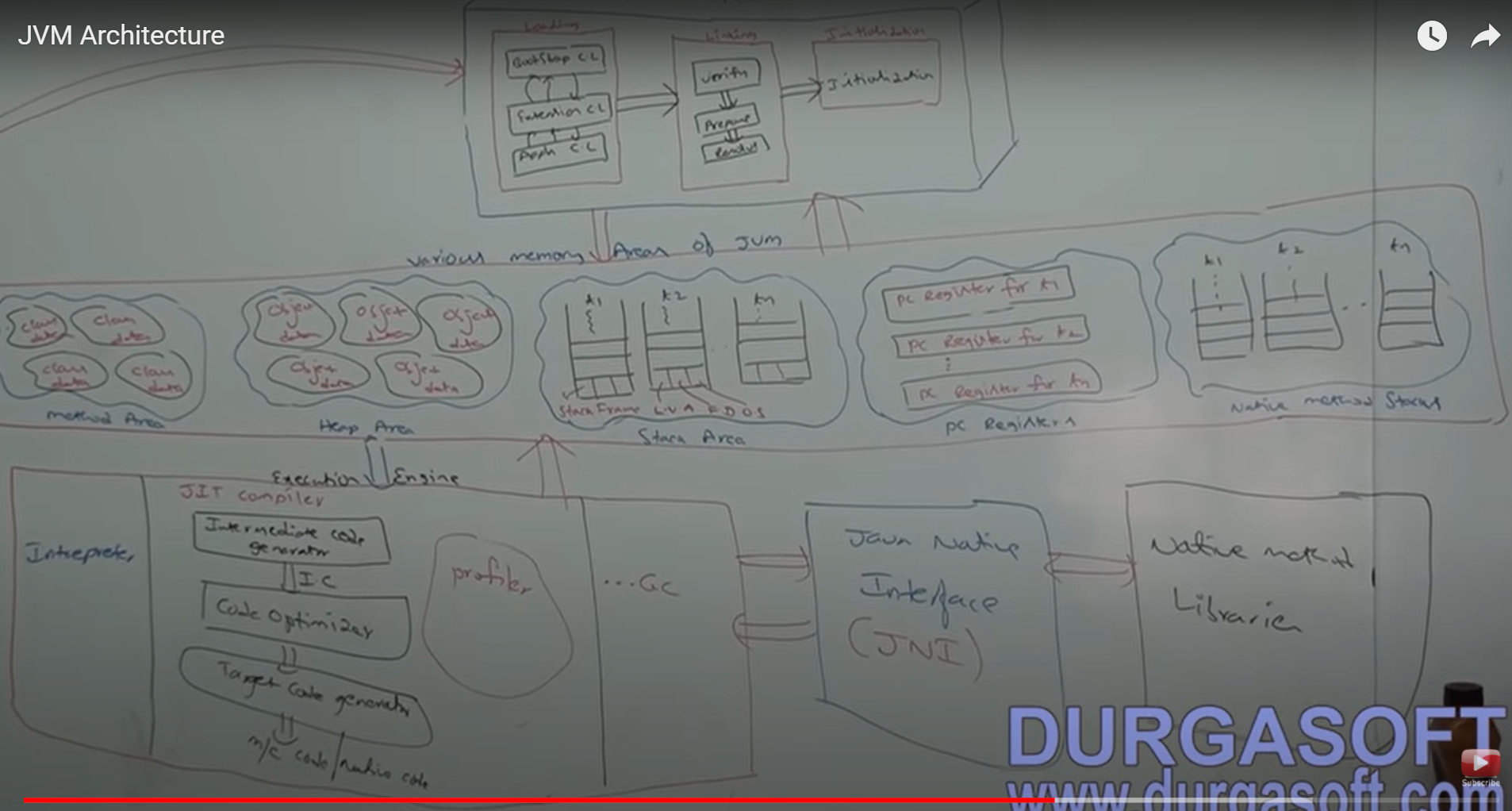
**Initialization:-** All original values will be initialized to the static variables.

**Various Memory Areas of JVM:-** 5 Types of Memory Areas are available inside JVM.

1. Method Area:- Class Level Data will be there. Static methods will be dumped here. One method area for One JVM
2. Heap Area:- Object Data will be there. Instance Variables are stored in Heap Area. One Heap area for One JVM
3. Stack Area:- All local variables are stored in Stack. For Every Thread Separate runtime Stack will be created etc., t1, t2,t3.. tn. Each entry is called Stack frame. Each Stack frame divided into 3 parts. This is **Thread Safe.**
4. Local Variable Array
5. Frame Data:- It contains all symbols. If any exception occurs data will be available in Frame Data.
6. Operand stack:- Intermediate operations to perform. It acts as a runtime.
7. PC Registers:- To hold address of Current Executing Instruction. PC register for Thread 1, Thread 2
8. Native Method stacks: For Every thread a separate runtime stack created to hold the native method information.

**Execution Engine:-**It has multiple components

1. Interpreter:- It interprets, compiles and executes
2. JIT Compiler:- for repeated method execution
3. Intermediate Code generator:- It produces the intermediate code
4. Code Optimizer:- It is responsible to optimize the code
5. Target code Generator:- it is responsible to generate machine code or native code.
6. Profiler:- It is responsible for hotspots. It is responsible to check method is repeated or not.
7. Garbage Collector
8. Security manager
9. Native method Libraries:- JNI(Java Native Method Interface)

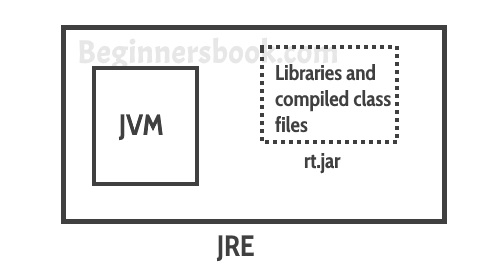


**JRE VS JVM AND JDK:-**

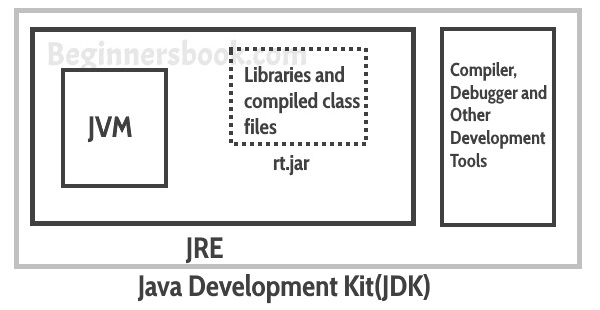
**JRE:-**JRE is an environment within which JVM runs. It contains JVM, class libraries and other files excluding development tools such as compiler and debugger.

We can run the code in JRE but we cant develop and compile the code in JRE.

**JVM**:- java Virtual Machine which converts the byte code into machine language code by using compiler and interpreter.



**JDK:-**  It is a superset of JRE. It consists of everything that JRE consists along with the Development tools compiler and debugger.



Explain simple Java Program explain public static void main(String args[])

**Variables in java**:- A variable is a name which is associated with a value that can be changed. Naming convention.

1. It should start with small letter only.
2. $,\_ we can use for variable names.

Three types of variables available in java

1. Local Variables
2. Static variables
3. Instance variables

Static Variables:- Class level variables. Common for all instances of the class.

Instance Variable:- Each instance has its own copy of variables.

Local Variable:- These variables are declared inside the method of the class. Scope is limited to that method.

**Data Types in Java:-**Data type defines the values that a variable can take, If a variable has int data type, it can only take integer values.

Primitive Data Types**:**- Eight primitive data types Boolean, char-1, byte-1, short-2, int-4, long-8, float-4 and double-8.

Literals in java**:**- A literal is a fixed value that we assigned to a variable in a program.

**Operators in Java:-** An operator is a character that represents an action.

1. Basic Arithmetic Operators
2. Relational operators
3. Bitwise operators
4. Auto-increment and Auto-Decrement Operators
5. Assignement

**SOLID OOPS Principles:-**

1. **S**ingle Responsibility Principle:- A class Should only have one responsibility. It helps to Test, Low coupling- Dependencies are less,Organization- Smaller and Well organized classes are easier to search.
2. **O**pen/Closed :- Open for Extension and closed for Modification.

publicclassGuitar **{** private **String make;** private **String model;** privateint **volume; //Constructors, getters & setters }** in future if we want to do some change in the Guitar class Instead of doing changes in the existing class**. Public class Flame extends Guitar{**

**}**

1. **L**iskov substitution:- Subclasses should be subsuitable for their base classes.

Vehicle is a base class and Car and Truk are subclasses. Lets suppose if we write one method called which is in Vehicle class getDriver(); it will be ambiguity either truck or car Driver so we need to specific at that time.

1. **I**nterface Segregation:- Large Interfaces should be spilt into small interfaces. So implemented classes will only focus on the methods those are required not every method unnecssarly.
2. **D**ependency Inversion:- Decoupling of software modules. Instead of depending on high level modules on low level modules both will depend on the abstractions

public class Windows98Machine {

private final StandardKeyboard keyboard;

private final Monitor monitor;

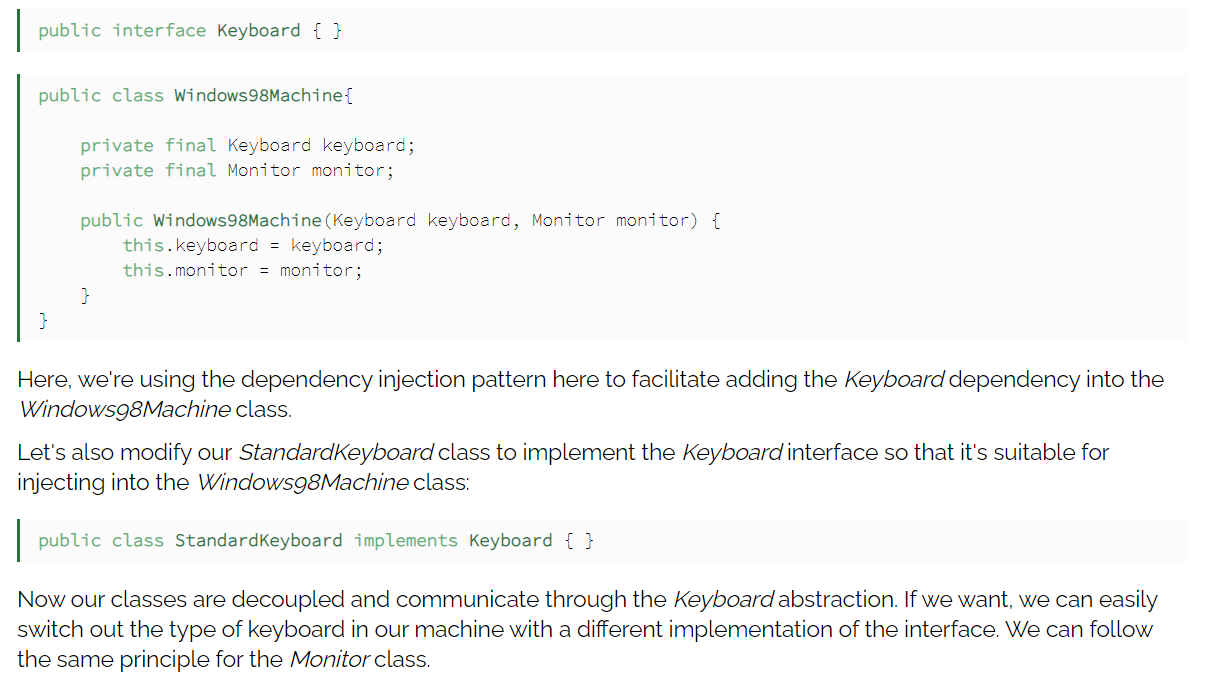
public Windows98Machine() {

monitor = new Monitor();

keyboard = new StandardKeyboard();

}

}



**Multi Threading in Java**

A Thread is a smallest part of a process that can run concurrently with the other parts(other Threads) of the same process. Threads are independent each other that’s is the reason if exception occurs in one thread that does not effect the other execution of other threads. All threads share the common memory. The process of executing multiple threads is called as multithreading. The main purpose of the multithreading is to provide simultaneous execution of two or more parts of a program to maximize the utilization of CPU time.

**Thread Life Cycle:-** A new Thread can be part of any of the following state.

1. New—A thread has not yet started.
2. RUNNABLE--- A thread is executing in the java Virtual Machine.
3. BLOCKED – A thread that is blocked waiting for a monitor lock.
4. WAITING – A thread that is waiting indefinitely for another thread to complete the task.
5. TIMED\_WAITING --- Waiting for another thread to execute the task upto a specific time.
6. TERMINATED:- A thread that is exited in this state.

**Multitasking vs Multithreading vs Multiprocessing vs Parallelprocessing**

Multitasking:- Able to execute more than one task in nothing but multi Tasking.

Multithreading:- Executing multiple threads simultaneous .

Multiprocessing:- Same as multitasking. Ability to use more than one CPUs in a task.

Parallelprocessing:- It refers to the utilization of multiple CPUs in a single computer System.

**Creating a Thread:-** Two ways to create a thread

1. Extending a Thread Class
2. Implemeing an Runnable Interface.

**Methods in Thread Class:-**

1. getName()- Obtaining a thread name
2. getPriority() – Obtain a thread’s priority
3. isAlive() – Determine a thread is still running
4. join() – wait for a thread to terminate.
5. run() – Entry point to run a thread.
6. sleep() – Suspend a thread for a period
7. start() – start a thread by calling its run method.

**Collections:-**

**ArrayList:-** ArrayList is a class which implements the List Interface. It internally follows the Array Datastructure. It dynamically grows and shrinks. Arraylist is not thread safe so we cant use the ArrayList in multithread environment.

ArrayList<String> list=new ArrayList<String>();

list.add(ele1);

list.remove(ele);

we can use the set method to change the element in ArrayList

list.set(3,newEle); import java.util.\*;

class JavaExample{

public static void main(String args[]){

ArrayList<String> alist=new ArrayList<String>();

alist.add("Steve");

alist.add("Tim");

alist.add("Lucy");

alist.add("Pat");

alist.add("Angela");

alist.add("Tom");

//displaying elements

System.out.println(alist);

//Removing "Steve" and "Angela"

alist.remove("Steve");

alist.remove("Angela");

//displaying elements

System.out.println(alist);

//Removing 3rd element

alist.remove(2);

//displaying elements

System.out.println(alist);

}

}

Iterate in an ArrayList:-

for(String ele:list){

Syso(ele);

}

**Size:-**list.size();

**Sorting:-**Collections.sort(list);

list.indexOf(element);

list.get(i);

al.subList(1, 4)

al.lastIndexOf(1)

There are two ways to synchronize ArrayList:-

Collections.synchronizedList()----Collections.synchronizedList(new ArrayList<String>());

CopyOnWriteArrayList:- It is a thread safe variant of ArrayList.

CopyOnWriteArrayList<String> list=new CopyOnWriteArrayList<String>();

**Swap Elements in ArrayList**:-Collections.swap(list,1,4);

**Serializable:-** ArrayList is by default serializable. To Serialize an object means to convert its state to a byte Stream so that byte Stream can be reverted back to the Object.

Clone:ArrayList<String> al2 = (ArrayList<String>)al.clone();

**Increase the Capacity of ArrayList:-** list.ensureCapacity(5);

Convert the LinkedList to ArrayList:-

LinkedList<String> linkedList=new LinkedList<String>();

ArrayList list= new ArrayList<string>(linkedList);

Convert the Array to ArrayList:-

ArrayList<String> list=new ArrayList<String>(Arrays.asList(array));

**ArrayList vs Vector:-** Both follows Array DataStructure. ArrayList is non-synchronized means multiple threads can work at the same time. But Vector is an Synchronized so single thread can work at a time.

Resize:- ArrayList half size vector double the size.

Fail-Fast:- ArrayList fail ast but not vector

Performance:- Arraylist is faster than Vector.

**Similarities:-**

1. Both follow Array DataStructure
2. Grow dynamically and Shrink
3. Follow insertion order
4. Allows duplicate and null values.

Java 1.8 Features

**Stream API** :- Stream API provides the functional programming to the java.

Advantages of Stream API:-

1. More efficient java programmer
2. Heavy use of lambda expressions.
3. ParallelStreams are used to work on the multi threading concepts

A stream pipeline consists of a source and zero or more intermediate operations and one Terminal operation.

Source--🡪Filter-🡪 sort --🡪 Map-🡪 Collection.

Stream source can be created from collections list, sets, long

Stream operations are either intermediate or Terminal.

Intermediate Operations:- such as filter, map or sort return a stream so multiple intermediate operations.

Terminal Operation:- such as forEach, void or even return a non stream value.

Intermediate Operations:-

1. anyMatch();
2. distinct();
3. filter()
4. findFirst();
5. flatMap()
6. map();
7. skip()
8. sorted()

One Terminal Opertion:-

forEach()

collect will collect all the elements into a collection

other options reduce the stream to a single summary element

1.count()

2. max()

3. min()

4. reduce()

5. summaryStatistics()

1.**Integer Stream**:- IntStream.range(1,10).forEach(System.out::print);

System.out.println();

2.**Integer Stream with skip:-** IntStream.range(1,10).skip(5).forEach(x->Syso(x)); //it will print 6,7,8,9

3**. Integer Stream with sum**:- IntStream.range(1,10).sum());

4. **Stream.of, sorted and findfirst function**

Stream.of(‘Ava’,’Aneri’,’Alberto’).sorted().findFirst().ifPresent(System.out::println);

1. Stream from Array, sort,filter and print

Arrays.stream(names).filter(x->x.name.startsWith(“s”).

Sorted().forEach();

1. Average of Square of an int Array

Arrays.stream(array).map(x->x\*x).average().ifPresent(Syso);

Stream of List filter and print

People.stream().map(x->x.toLowerCase()).filter(x->x.startsWith(‘a’)).forEach(Syso);

1. Stream rows from text file sort filter and print

Stream<String> bands=Files.lines(Paths.get(“bands.txt”));

Bands.sorted().filter(x->x.length()>13).forEach(syso);

Bands.close();

1. Stream of rows from test file filter, collect as list.

Bands.filter(x->x.contains(“a”)).collect(Collections.sort())

**Java 9 Features:-**

1. Jigsaw
2. REPL—Read Eval Print and Loop
3. Jshell
4. Enhancement in Stream API by adding some methods
5. Modulrity--- rt.jar used to work with multiple classes. Now they have introduce modules to reduce the load on rt.jar
6. HTTP2.0 in progress

It will used to learn the java

It is user friendly

Jshell>int num=5;

2>num==5

$2

5

/exit

Goobye

/>

Autocompletion:-

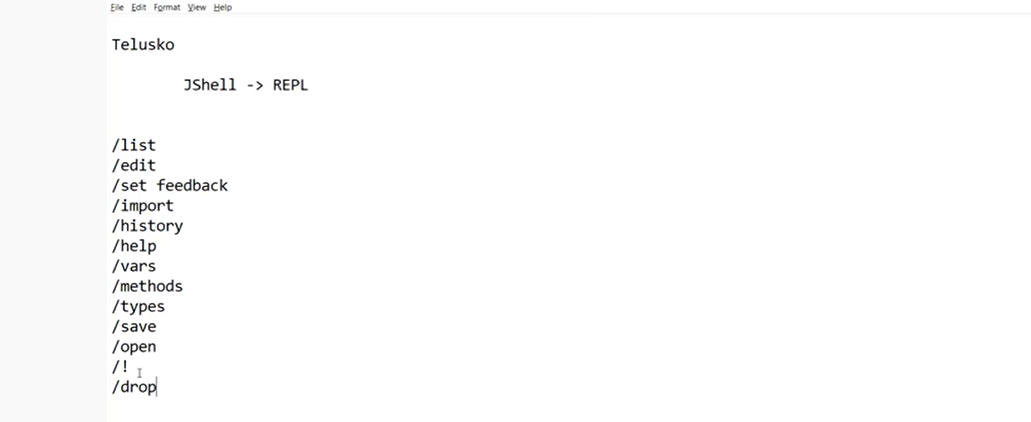
Jshell>/vars :---- Gives the variables we have used so far

/list

/import

/history

/help



**Modularity:-**Java uses the rt.jar which consists or more classes. Each and every new version rt.jar size is getting increased it might be a problem in future. So modularity comes into picture. If we don’t want specific classes like RMI. Why do we need to include in my project.

